Development of a Parallel Discrete Event Simulation
Transactions vs. Data Resources

Gary A. Allen

TRW Data Technologies Division

730 Irwin Avenue, Room 270
Schriever Air Force Base, CO 80912-7300
gary.allen@jntf.osd.mil

ABSTRACT

Modeling a Parallel Discrete Event Simulation to provide research into the execution of a war game over multiple processors presents many challenges. Next generation war games are likely to execute over a billion events, requiring several days to complete on a single processor. Through the use of multiple processors and advanced time management algorithms, the goal to run war games much faster than real time is a reality. This paper provides an overview of research at the Joint National Test Facility using SES/*workbench*® to model a war game distributed over multiple processors. A brief history of preliminary models is given. Comparative results of these different models are given which show the accuracy of the preliminary models. Two models were then adapted to emulate the anticipated computing load for War Game 2000 and the timing algorithms used in the Synchronous Parallel Environment for Emulation and Discrete Event Simulation framework that are the main focus of this paper. One model described is transaction based and the other is data resource based. Advantages and disadvantages of the two modeling paradigms are discussed throughout the paper. Specific techniques used in these models are described in detail and show the adaptability of the sequential SES/*workbench* tool to model a parallel discrete event simulation. We have modeled the current command and control simulation using detailed data on processor and communications resource usage. A model of the new war game under development has also been constructed and calibrated. Time management algorithms have been prototyped. This "sim of a sim" is poised to reduce risk for development of current and future war games.

Introduction

The Parallel Discrete Event Simulation (PDES) environment presents many challenges in modeling. The Joint National Test Facility (JNTF) is well known for its war game facilities. The Technology Insertion Studies and Analysis (TISA) group has been tasked to support War Game 2000 (WG2K) development. One task is to model the next generation war games with SES/*workbench*® (SESWB) by emulating the message traffic (several hundred thousand to possibly a billion messages) over multiple processors without violating causality. A second task is to use the SESWB tool to determine what hardware will provide the best performance for WG2K. We are currently building WG2K using the Synchronous Parallel Environment for Emulation and Discrete Event Simulation (SPEEDES) framework to control the synchronization process. SPEEDES uses the *Breathing Time Warp* algorithm to achieve this synchronization. Breathing Time Warp is a combination of two optimistic synchronizing algorithms, *Time Warp* (Jefferson 1985) and *Breathing Time Buckets* (Steinman 1991).

Preliminary steps to building a PDES model included building a calibrated Advanced Real-time Gaming Universal Simulation (ARGUS) Model, a parameterized Threat Load Model, an aggregated WG2K Model, and a SPEEDES Model. The ARGUS Model showed that an existing war-game scenario could be modeled accurately (within 10%) with SESWB. The Threat Load Model demonstrated how events could be spawned from other events (i. e. missiles dropping boosters or launching Re-entry Vehicles (RV's)). The aggregated WG2K Model provided a representation of a not yet existing WG2K at the early block 10 stage. The SPEEDES Model was constructed to allow full representation of SPEEDES queuing and event synchronization algorithms.

By modeling WG2K with the SPEEDES framework along with other hardware architectures, we have the opportunity to measure performance speedup without running tests on the actual hardware represented in the model. The purpose of this paper is to describe the progress and specific techniques used to model the WG2K message traffic and the SPEEDES framework with both the transaction and data resource based paradigms.

Methods

The initial effort was to emulate the message traffic generated by an ARGUS scenario with a Commercial Off the Shelf (COTS) product, SES/*workbench*®. Statistics were captured from the ARGUS run and used as calibration data for the SESWB and Thread Builder (TB) ARGUS models. Comparative statistics included number of missiles, re-entry vehicles, interceptors, and the frequency and number of messages sent between objects. The ARGUS Model provided an opportunity to learn some of the capabilities of SESWB (multi-module model, routing, etc.) and techniques that became very valuable for the eventual WG2K Model.

A Threat Load Model then provided an opportunity to see how the load expands with different scenario combinations. Again, SESWB and TB models were created to compare with each other and the Excel® scenario spreadsheet listing objects and timing predictions. The SESWB model is parameterized to allow for back-to-back runs of different scenarios without requiring the model be rebuilt for each run. This model has been implemented as the threat load object for the aggregated WG2K Model.

Creation of the SESWB Aggregated WG2K Model allowed for a first look at potential bottleneck problems such as the Host Router and gateways. The Threat Load Model was re-used as the main module for the multi-module WG2K Model (**Figure 1**). The model is segmented to represent the various numbers of instances for missile and radar objects. SESWB allows great flexibility for creating multiple instances of an object (or submodel) and passing on changes made in the original instantiation of the object. All of the instances inherit those changes when the model is built, thus saving much time when changes are made. Message passing between separate modules is necessary to allow greater flexibility and provides a more realistic representation of objects outside the main simulation module (MADSIM) for the projected WG2K.

Since a typical war game includes much human interaction, the simulation models an estimate of probable responses and the time anticipated for making those responses. In the current WG2K model, the Battle Management Command, Control, & Communications (BMC3) initiates many messages on a periodic basis and some on an irregular basis to emulate human input. The irregular messages are usually responses to the spawning of different scenario events such as missile launches. In SESWB, this requires a broadcast of these events to all submodels throughout the model since some submodels message number or size depended on this information.

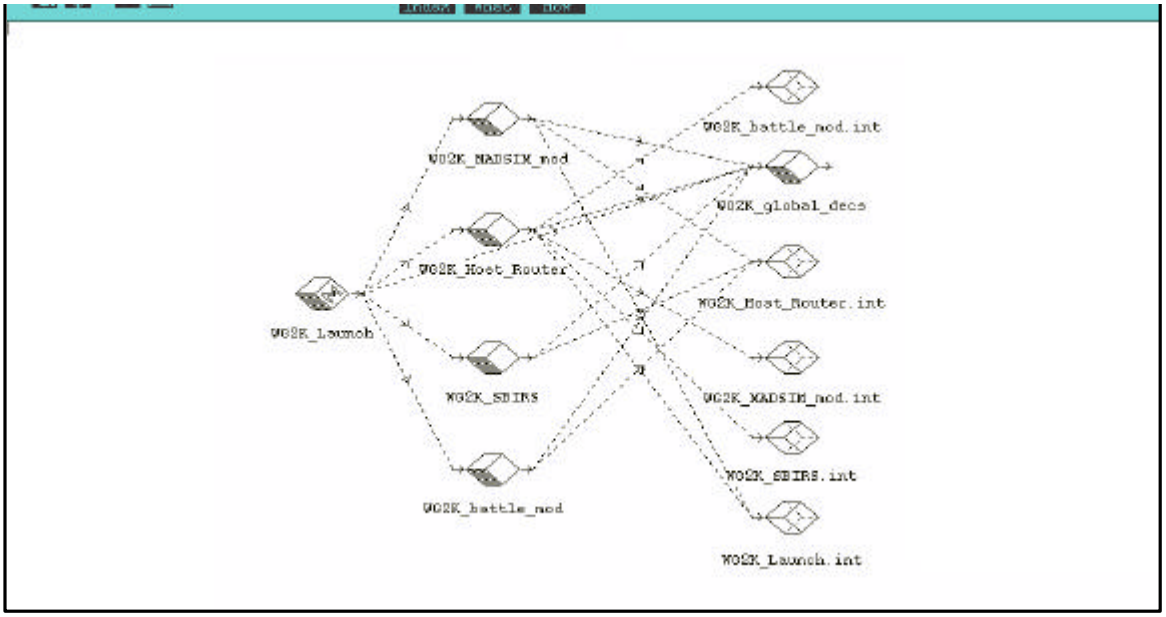

Figure 1: The Aggregated WG2K Model - The WG2K_Launch main module was a copy of the Threat Load Model with additional functionality added. Space-Based InfraRed System (SBIRS), Battle Management Command, Control & Communications (BMC3 - battle_mod), and the Host Router were in separate modules to show that they were not a part of MADSIM which represents all sensors.

The major problem associated with the WG2K model is the routing of messages to specific instances of objects. Being a multi-module, multi-instance model, this requires some advanced techniques to send information to or retrieve information from distant submodels. Some messages are sent through a host router (**Figure 2 & 3**) or a gateway to all instances of an object, while other messages are only be sent to a specific instance of an object. All of these problems are associated with the war game environment here at the JNTF. Various enhancements have been implemented as a result of these early models.

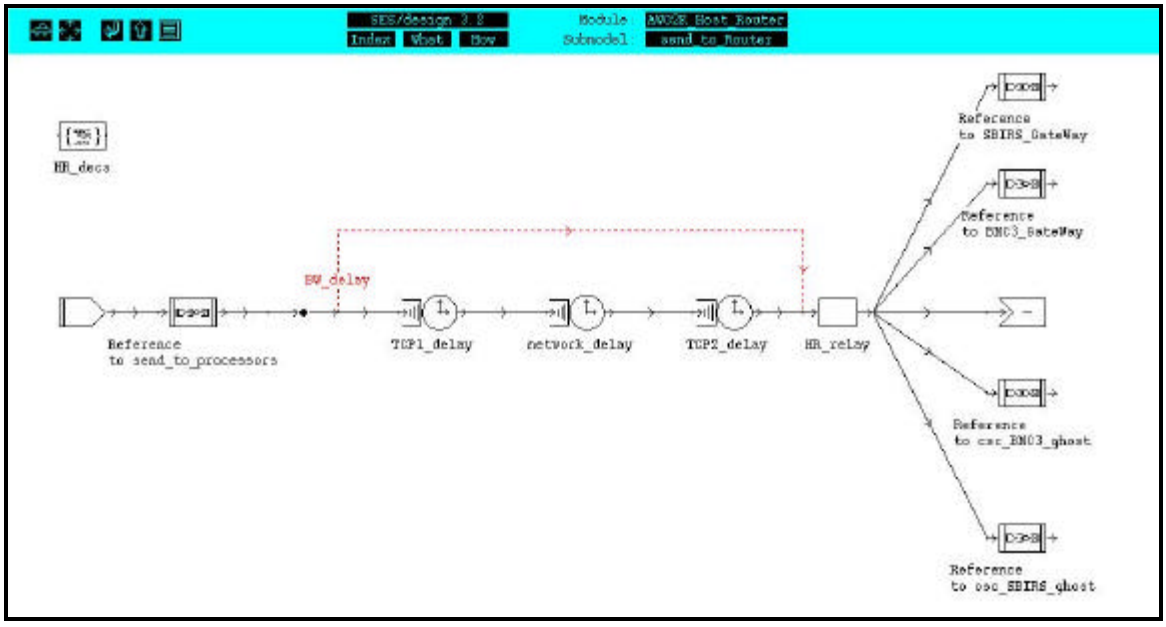

Figure 2: Host Router (transaction based)– Representation of the host router and network communications.

Note that the data resource model (**Figure 3**) transfers the message to the processor (tempQ resource) and then receives the processed message at get_HR_proc_in_msgs. The rest of the sequence is the same except the data resources are transferred into a resource box instead of directly routing the transaction.

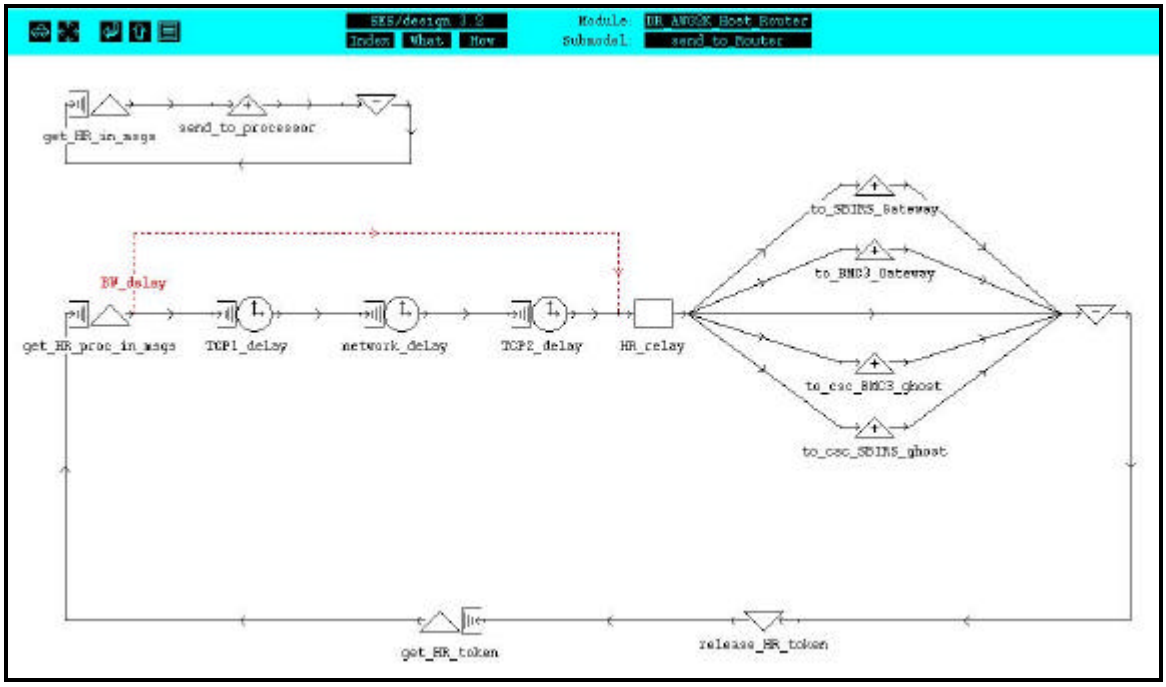

Figure 3: Host Router (data resource based)– Representation of the host router and network communications.

Since WG2K has chosen the SPEEDES framework to distribute our war game over multiple processors for maximum speedup, a separate SPEEDES Model was built to allow for the different queuing and timing methods incorporated within SPEEDES. Modeling of the SPEEDES framework proved to be quite challenging due to the queuing and timing algorithms used within SPEEDES. Following is a brief description of the more important parts of SPEEDES modeled with SESWB.

SPEEDES assigns object processes to processors at the beginning of the simulation. A SESWB card deal model was developed to realistically represent this function. To represent the same distribution as a Thread Builder run, one must be certain that the objects are assigned in the same order as the spreadsheet input used with Thread Builder. Note that the only balancing used here is the number of objects per processor. No load balancing occurs due to number of messages or message sizes being sent.

The parallel computing strategies used in SPEEDES begin with the Time Warp algorithm. Time Warp uses the *virtual time*[3] paradigm first developed by Jefferson in 1985. This paradigm represents an optimistic approach to organize and synchronize distributed systems. Since causality (you can't send a message in the past) is very relevant in war gaming, this presents a problem of major importance. The system can not be allowed to violate causality, but should be allowed to run ahead of wall clock time. This paradigm uses anti-messages to rollback time and cancel messages that were processed and sent optimistically with a time ahead of Global Virtual Time (GVT) that corresponds to the earliest time stamp of any message in the simulation.

To keep the separate processors from "running away" from the rest of the system, two timing elements are used. A global clock (**Figure 4**) keeps the systems GVT and each processor keeps track of it's own Local Virtual Time (LVT). SPEEDES uses a set of flow control variables to notify the system when it is time to update GVT. GVT will be updated periodically if a certain amount of time (tgvt) has passed without a GVT update caused by some other factor. Keeping track of messages (events) processed also causes GVT updates. SPEEDES allows all messages with a time stamp less than GVT to be processed and sent

immediately. A set number of Nrisk messages (messages with a time stamp > GVT) are also allowed to be processed, but a copy of these messages is kept until a GVT update occurs. After a set number of messages are processed and sent from a processor, that processor notifies the system that it thinks GVT should be updated. After all processors have voted to update GVT, the system interrupts all processors and waits for all sent messages to be flushed from the communications network. The system then checks each of the processors' LVT and updates GVT to the earliest LVT. This meant that the SESWB model needs each processor instance keep track of its own set of variables.

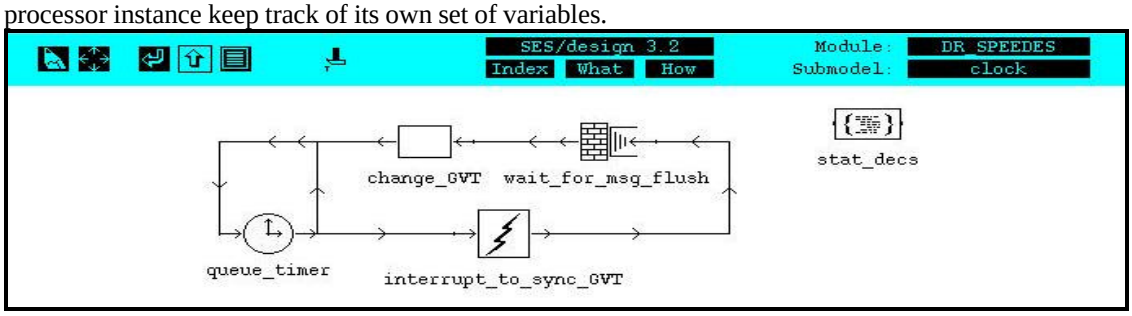

Figure 4: The Global Clock – This submodel periodically cycles through the queue_timer. If it is time to update GVT or all processors have voted to update GVT, the system: (1) interrupts all processing at interrupt_to_sync_GVT, (2) waits for all messages to be flushed from the communications system, and then (3) decides what GVT should be updated to (the earliest LVT) at change_GVT.

The queuing system used in SPEEDES also presents some unique problems. Each processor requires two ordered queues, a temporary queue and a Qheap. SPEEDES actually has a Qheap for each object assigned to the processor, but for simplicity sake, only one Qheap is in the current SESWB model. Both queues require a merge sort routine to order the messages by their time stamp upon arrival. SESWB makes this simple to create through the use of multiple instances of an object, but the communication of variable values in different instances does become quite complex.

The temporary queue (**Figure 5**) requires the ability to check each incoming message time stamp. Messages that are in the correct timing sequence are added to the end of the queue. Due to a requirement of SPEEDES to know the time stamp of the first message in the temporary queue, a double block is necessary to separate the first message from the rest of the queue. Incoming messages are blocked once a straggler message (an out of sequence message) arrives at the temporary queue that causes it to be reordered. A straggler message requires all messages in the queue be looped through in the tempQ_merge submodel (**Figure 6**) to determine the correct position to insert the straggler message. After the messages have been reordered the messages are reentered into the temporary queue and the blocked incoming messages are then released to enter the temporary queue.

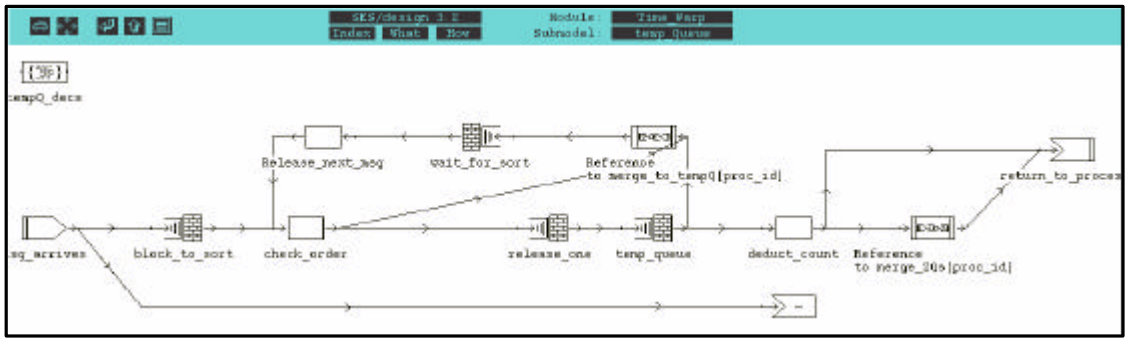

Figure 5 (transaction based): The Temporary Queue – An arriving message has its time checked at check_order. If the message has a time greater than the last message in the temporary queue, the message becomes the last in the queue. Otherwise, the message sets a block condition at block_to_sort and then proceeds to the merge_to_tempQ submodel along with all messages in the temporary queue (includes messages at release_one) to allow the new message to be merged into the correct position. Messages return to the temp_Queue submodel after they

are placed in the correct order and are blocked at wait_for_sort. After all messages have returned, they are then returned to the temporary queue and any message at the block_to_sort node is released sequentially.

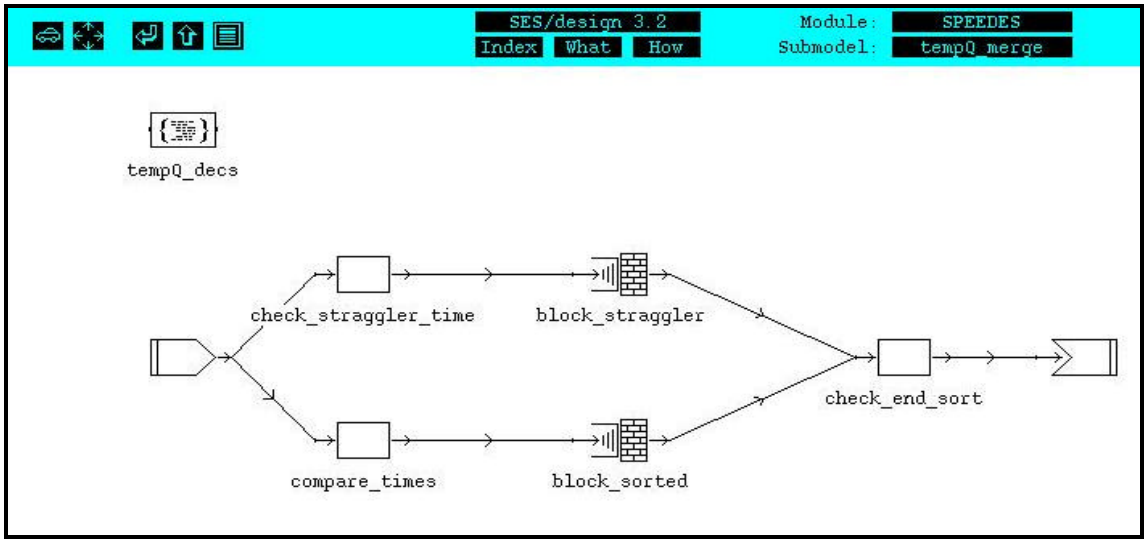

Figure 6 (transaction based): Merging into the Temporary Queue – An arriving straggler message has its time compared with messages already in the temporary queue. All messages are then returned to the temporary queue in the correct order by time stamp.

The two submodels in Figures 5 & 6 are combined in the data resource based model's temp_queue (**Figure 7**). Note that no blocking is required when a message first arrives in the submodel because there is only one transaction populated in the submodel at msg_arrives, which therefore automatically provides a blocking condition. The amount of coding for the data resource submodel is much less than the transaction-based submodel. This version of the temporary queue has the functionality of comparing the incoming message time stamp with that of the first message in the queue. If the incoming message needs to be placed in the front, all messages are moved down one position in the temporary queue without comparing each time stamp and the incoming message is placed at the front of the queue. The loop for inserting a message in the middle of the temporary queue avoids having to sort through the rest of the messages after the position for the incoming message has been found.

There are two conditions that result in the temporary queue being merged with the QHeap. The first condition involves a parameter for the maximum capacity set for the temporary queue size. Once this value is reached, all messages in the temporary queue are then merged into the QHeap. The second condition occurs if the temporary queue has been idle for a set amount of time that is monitored by the Global Clock. This passage of time also causes a merge of the two queues.

This is basically the same as the transaction based paradigm with the exception that a full temporary queue is added directly to the QHeap if the first temporary queue message time is greater than the last QHeap message time. Once again this avoids having to loop through all of the QHeap messages unless the two queues need to be merged. If the two need to be merged, the temporary queue is dumped into a temp_QHeap resource to merge together with the QHeap.

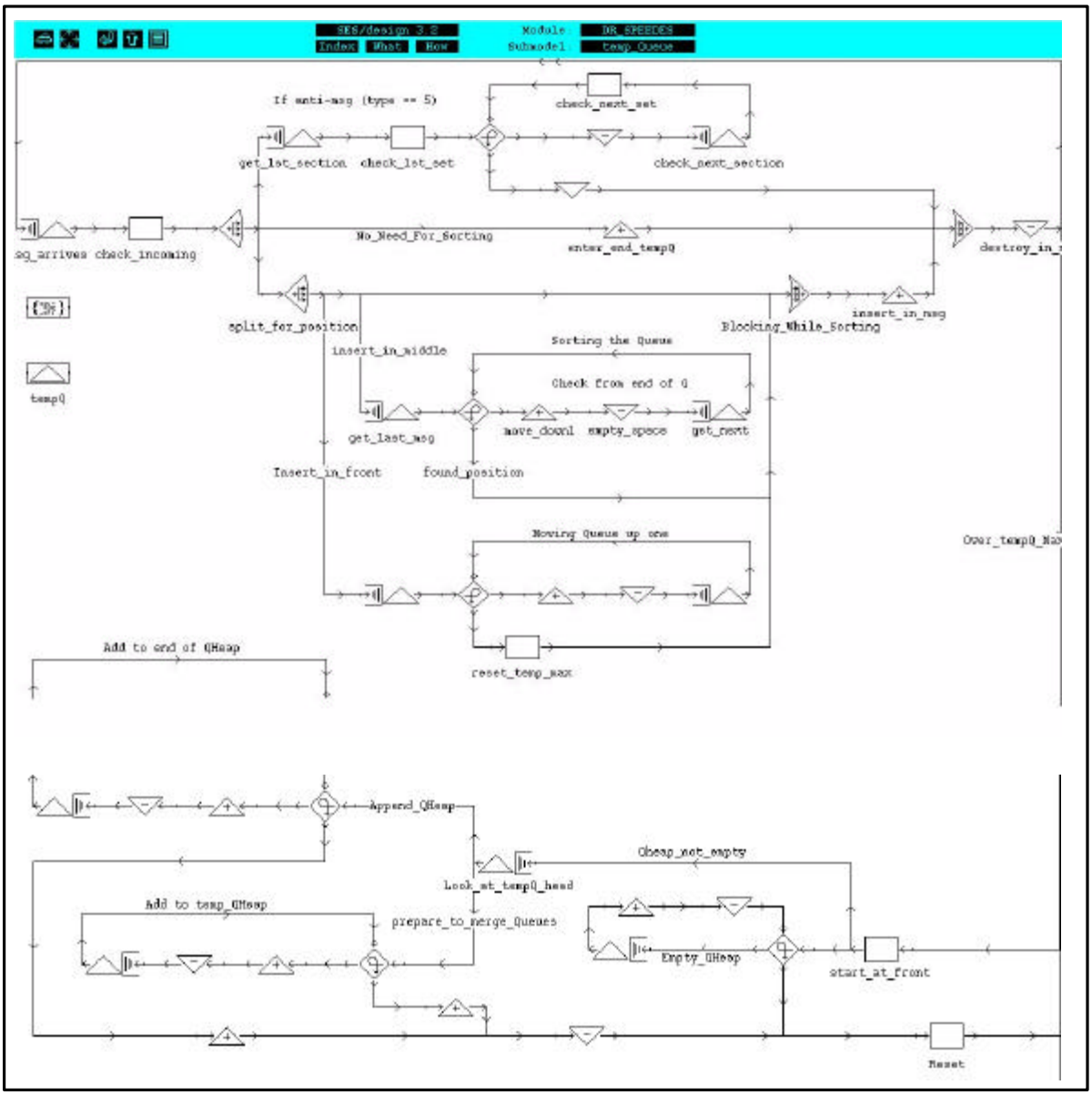

Figure 7 (data resource based): The Temporary Queue – Messages are entered into the temporary queue one at a time, either at the end, in the middle, or at the front. Once the temporary queue is full, those message are added either to the end of the QHeap or placed in a temp_QHeap resource to be merged with messages in the QHeap.

Once the temporary queue is full, the temporary queue messages must be merged into the correct order with the QHeap (**Figure 8 & 9**) messages to wait for processing. Messages are released to the block_tempQ or block QHeap nodes one at a time. The earlier time stamped message then advances to the wait_for_finish node. This process is repeated until all messages have been merged together. A slight delay is implemented before the messages are returned to the QHeap. Once again a double block mechanism (hold_queue and wait_for_merge) was required to allow the first message of the queue to be separated from the rest of the messages in the QHeap. The data resource model has the advantage of reusing the main body of the temporary queue with a change for the data resources being accessed.

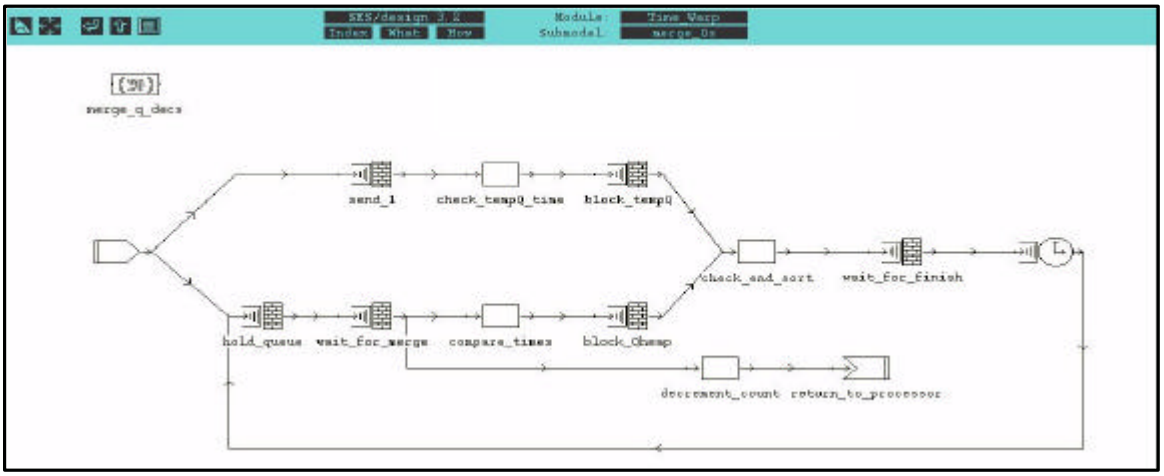

Figure 8 (transaction based): The QHeap – The merging of the two queues requires a comparison of one message from each queue at a time. The earlier message is released at either block_tempQ or block_Qheap. The first message in the Qheap waits for the processor at the wait_for_merge node.

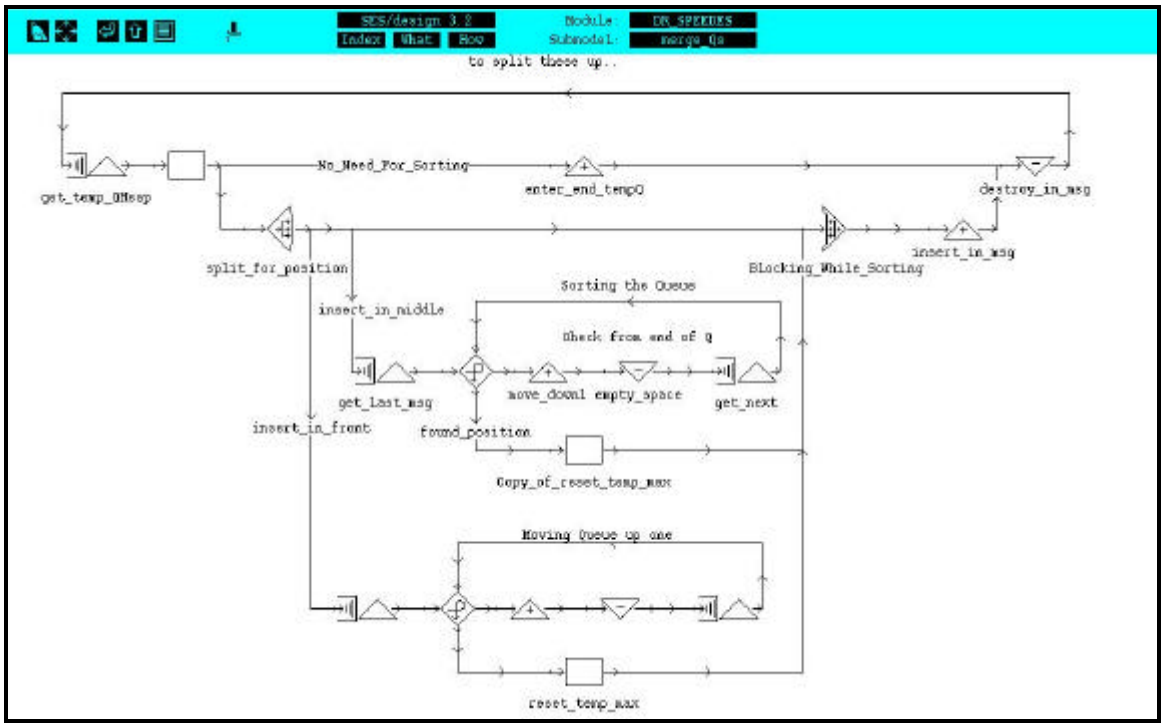

Figure 9 (data resource based): The QHeap – The merging of the two separate queues provides reuse of the same sequence used in the temporary queue uses.

When a processor becomes available, it needs to allow the next ordered message to enter either from the temporary queue or the QHeap. This situation presents an interesting problem because the processor needs to check both its temporary queue and its QHeap to decide which message had the earliest time stamp. A double block mechanism (release_one and temp_queue for the temporary queue) is required in SESWB to solve this problem in the transaction-based paradigm. By separating the first message in each queue from the rest of the queue, the processor only looks at the first message time of each queue (since the queues are already ordered) and decides which message should access the processor next. After the selected message enters the processor, the queue that released that message must move the next ordered message up to the first block node.

SPEEDES uses flow control parameters to determine the phase that Breathing Time Warp is in (**Figure 10**). The model functions in normal mode when messages arrive at the processor with a time stamp less than GVT. When messages arrive with a time stamp greater than GVT and the number of Nrisk_msgs is less than Nrisk_max (maximum number of optimistically processed and sent messages), SPEEDES enters the Time Warp phase. Once the number of Nrisk_msgs equals Nrisk_max, SPEEDES moves to the Breathing Time Buckets phase. Either after all processors have asked for a GVT update or a certain amount of time has elapsed, the GVT phase begins. All processors are interrupted and wait for all messages in transit to arrive at their destinations. Nrisk message copies are then cleared and then Nopt messages are committed and sent.

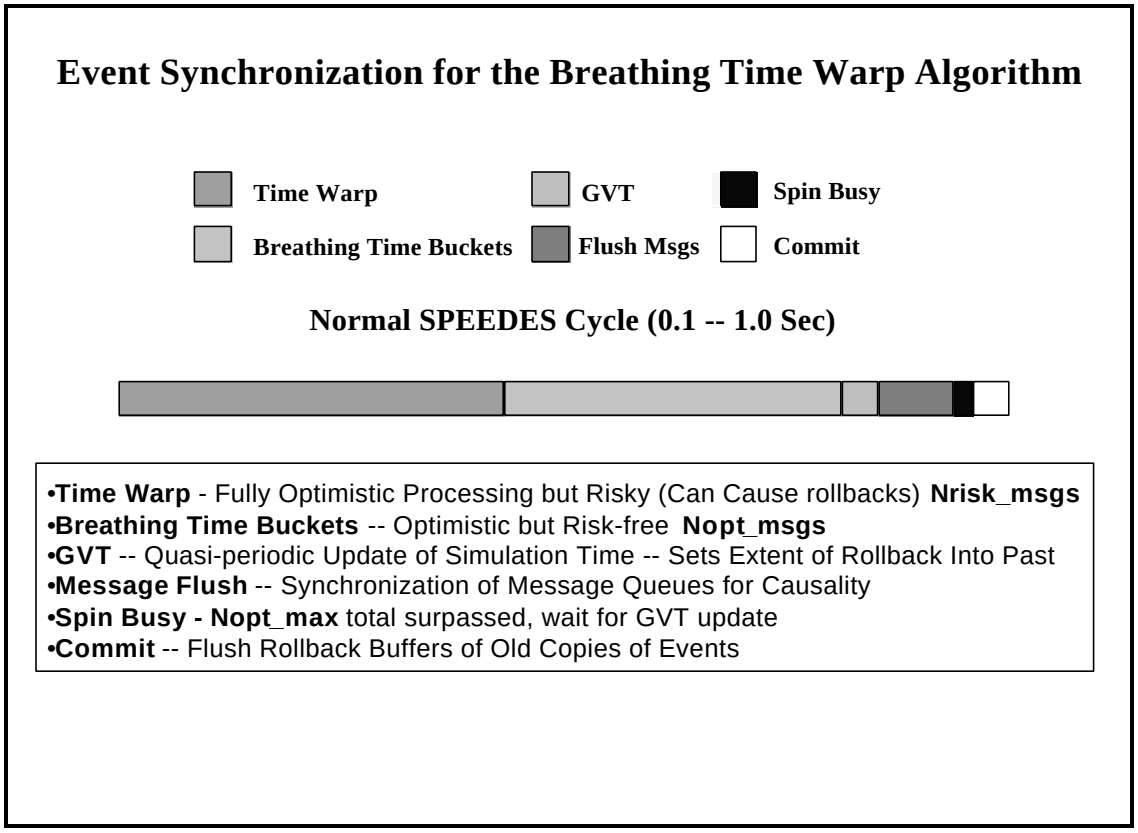

Figure 10: Event Synchronization for the Breathing Time Warp Algorithm

Messages routing in the processor (**Figure 11 & 12**) show additional paths besides those that are determined by the Breathing Time Warp phase. Since copies are only needed for outgoing messages, the incoming csc messages are processed and returned without copies being made. Each anti-message is sent to the processor that received the original optimistically processed message and labels that message canceled in the temporary queue, QHeap, or the processor where it resides. That message is then removed from the system when it tries to access the processor.

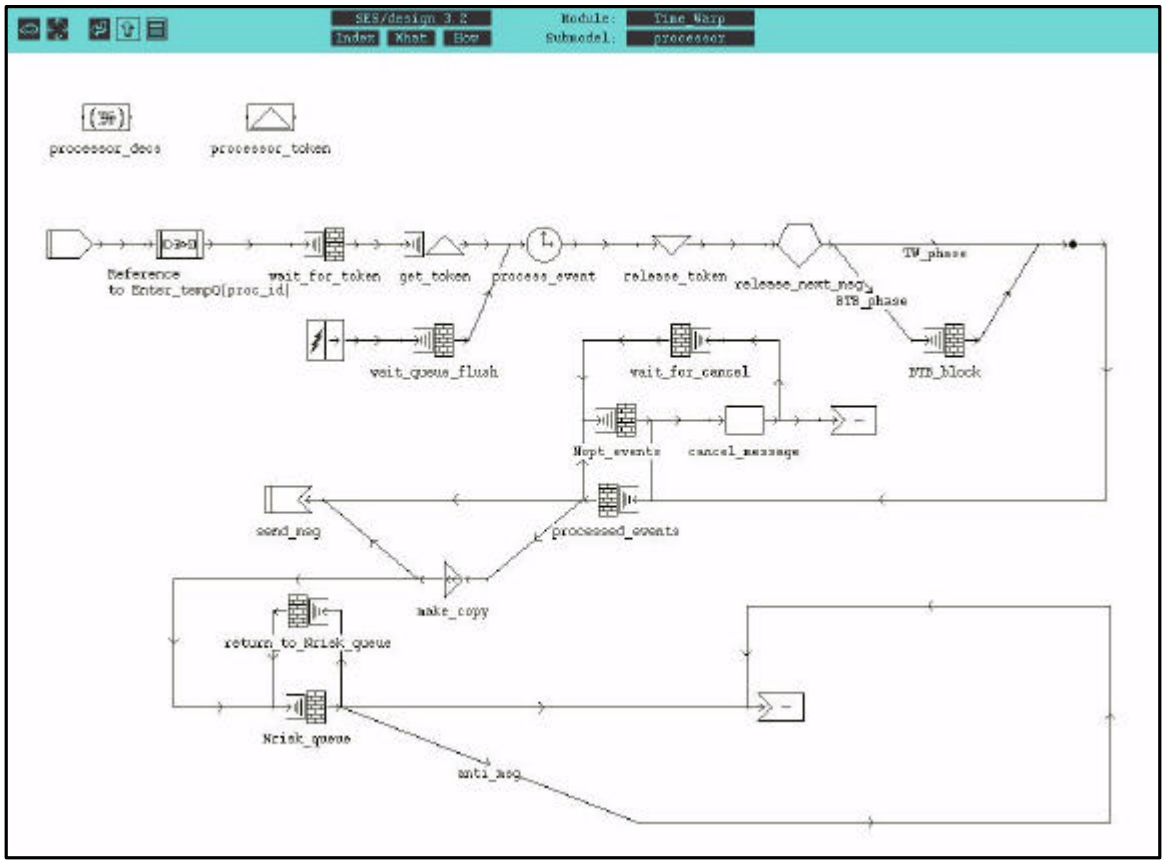

Figure 11 (transaction based): The Processor Node – Messages are selected from the temporary queue or QHeap by their time stamp order if a resource token is available. Messages enter the processor, receive the resource token, and are then processed at process_event. If a GVT update occurs, the message is interrupted and then blocked until the GVT is updated. An uninterrupted message leaves the process_events node, releases its token, and checks the first messages waiting in each queue at release_next_msg. The earlier of the two messages are then released to enter the processor. These messages then have three possible paths to take, send messages that have a time stamp earlier or equal to GVT, make a copy and send messages if the Nrisk_count is less than Nrisk, or store messages in Nopt_events to wait for a GVT update. If a straggler message arrives at the processor, an anti message is sent or a message at the Nopt_events node is cancelled. The time stamp for the first message at Nrisk_queue is stored to determine if an anti-message needs to be sent or a message at Nopt_events needs to be cancelled. If the straggler message time is less than the first Nopt_events time, the Nrisk_queue is looped through to send an anti-message for any message sent with a time stamp greater than the straggler message time. If the straggler message time is larger than the first Nopt_events time, the Nopt_events are looped through to cancel any messages with a time stamp greater than the straggler message time. NOTE: This model does not actually send anti-messages! It is only simulated they are sent.

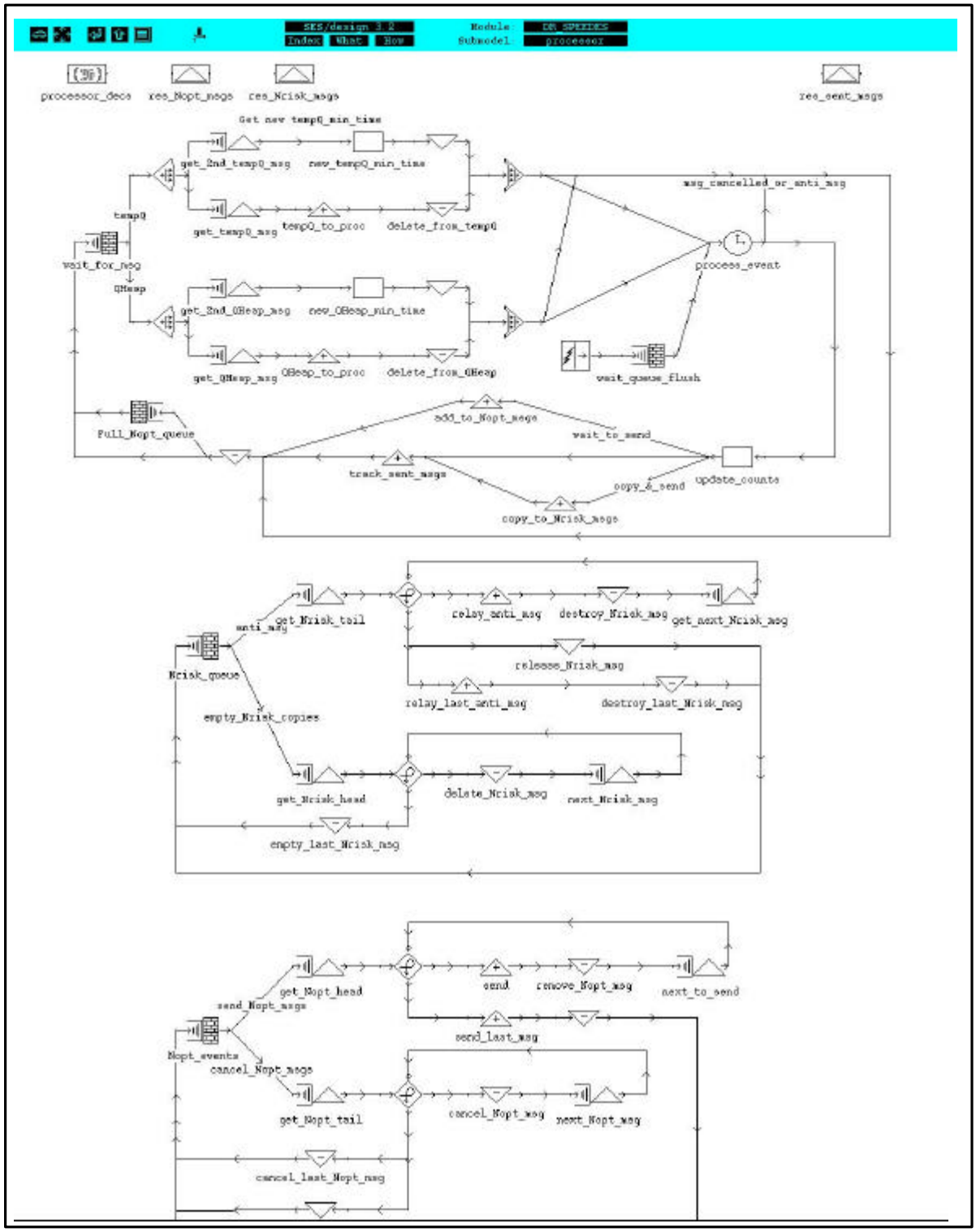

Figure 12 (data resource based): The Processor Node – The next message is selected by comparing the time stamp of the first message in each queue. Both loops are identical except the resource being accessed (reuse again). If

the message is labeled canceled or as an anti-message, it is not sent on to another processor or copied. The bottom loops are again nearly identical. The Nrisk portion send anti-messages or empties messages from Nrisk after a GVT update. The Nopt portion cancels messages or sends them to the next processor after a GVT update.

Results

Both SESWB WG2K Models were calibrated to an Excel spreadsheet scenario and matched the results of the Thread Builder Model and the predicted results of the spreadsheet. Both models provide an accurate representation (within 10%) of predicted message traffic for WG2K as indicated by the current spreadsheet analysis (**Figure 13**). Message traffic was monitored during the simulation run to watch potential bottlenecks. A large part of the model is parameterized to allow greater flexibility in changing the scenario script at run time. The host router correctly relays messages between objects. The model of the communications network is parameterized to allow for inputs to represent different network architectures. A parameter is also included to allow for different speed processors.

CSC	Estimate	SES
BMS	64	64
RV	6,182	6,144
BMC3	216,706	222,360
SBIRS	2,659	2,724
IFICS	2,080	2,353
GBR	9,691	9,640
XBR	9,711	9,641
UEWR	291,300	287,346
***total**	543,167	545,146

Figure 13: Calibration Results – This table represents the number of messages sent by some of the CSC's (Computer Software Component) in WG2K. The estimate column was obtained from a spreadsheet analysis estimate and the SES column shows the results from the SES WG2K model. *NOTE: This total includes other CSC's not listed.

The SESWB SPEEDES Model accurately emulates the processes contained in the SPEEDES framework. Parameters were implemented to allow the same flexibility that exists within SPEEDES. The queuing theories are properly represented and both timing algorithms in *Breathing Time Warp*, *Time Warp* and *Breathing Time Buckets*, were correctly implemented in the SESWB model. Experiments were run on the separate SPEEDES model to validate the two causes for updating GVT, the passing of a limited amount of time without an update or all processors ask for an update). Message times were scrambled to cause straggler messages to arrive at the temporary queue and the processor. The queuing and interrupt functions worked as expected. The functions for sending anti-messages or canceling of messages not already sent due to the arrival of a straggler message both worked as expected.

Discussion

The ARGUS Model is a simulation of an already existing ARGUS simulation. Message traffic and the scenario timing were successfully modeled. Individual and overall message totals were calibrated to within 10% (most were within 1%) based on the profile data generated from an actual simulation scenario. The experience gained led to efficient simulation of the emerging WG2K scenario.

The WG2K Model represented the anticipated WG2K in advance of code development. This model was based on anticipated functionality, estimated lines of code, and operational message sets. It was calibrated to within 10% to the input spreadsheet and the results from the Thread Builder model.

Both the ARGUS and WG2K models allow us to demonstrate potential scalability to larger and faster processing power. Communications were modeled to show potential speedup due to faster communication links.

The innovation was using SESWB, a sequential modeling tool, to model time management algorithms for a PDES. In particular, the SPEEDES model represented the Breathing Time Warp algorithm used in SPEEDES.
Although both modeling paradigms were capable of achieving the same results, the data resource paradigm provide reuse of many concepts and much less coding required. Although memory usage (**Figure 14**) varied and does not appear to be a major factor, the transactions based model slowed considerably at sim time 1680. Another important item to note is the data resource model had much more functionality added. In particular was the addition of actually sending anti-messages in the data resource model that created an added load on message traffic. The transaction-based model only simulated sending the anti-messages. **Despite the added load, the data resource model was still 2.6 times faster than the transactions-based model!**

Transaction vs. Data Resource Statistics

Sim Time	Transaction-Based Model Size	Data Resource Model Size
260	13	15
280	14	15
764	17	20
1680	20	20
1700	25	20
1720	27	20
1780	28	31
1800	30	31
	Run Time = 51:16 min Took 1:06 to run 20 sim tics at one point!	Run Time = 19:35 min **2.6 times faster!!!**

Figure 14: Comparison Statistics

Acknowledgments

The Technology Insertion Studies and Analysis group at the Joint National Test Facility at Schriever AFB provided the research described in this paper. Reference herein to any specific product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Joint National Test Facility. I would especially like to thank Pat Talbot for his help with Threat Loading module, Ronald Van Iwaarden and Mitchell Peckham for assistance in modeling SPEEDES, Jeff Wehrwein for his help with Thread Builder, and Jim Eamon for his help with the calibration data.

References

[1] Steinman, Jeff S. ; Pasadena, CA "SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation". International Journal in Computer Simulation. Vol. 2, No. 3, January 1997, pages 251 – 286.

[2] Steinman, Jeff S. ; Pasadena, CA "Breathing Time Warp". In Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS 93). Vol. 23, No. 1, July 1993, Pages 109 – 118.

[3] Jefferson, David R. ; University of Southern California, " "Virtual Time." *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 3, July 1985, Pages 404 – 425.